

Attack Methodology Analysis: SQL Injection Attacks

Bri Rolston

September 2005



The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

Attack Methodology Analysis: SQL Injection Attacks

Bri Rolston

September 2005

**US-CERT Control Systems Security Center
Idaho Falls, Idaho 83415**

**Prepared for the
U.S. Department of Homeland Security
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517**

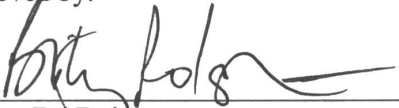
US-CERT Control Systems Security Center

Attack Methodology Analysis: SQL Injection Attacks

INL/EXT-05-00572

September 2005

Approved by:



Author: Bri Rolston
Cyber Security Researcher



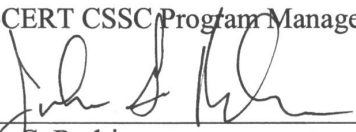
Date



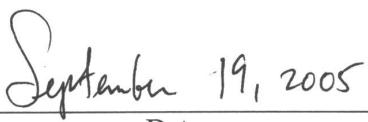
Fred C. Cowart
US-CERT CSSC Program Manager



Date



Julio G. Rodriguez
Program Lead, SCADA/Power Systems



Date



ABSTRACT

Database applications have become a core component in control systems and their associated record keeping utilities. Traditional security models attempt to secure systems by isolating core software components and concentrating security efforts against threats specific to those computers or software components. Database security within control systems follows these models by using generally independent systems that rely on one another for proper functionality. The high level of reliance between the two systems creates an expanded threat surface.

To understand the scope of a threat surface, all segments of the control system, with an emphasis on entry points, must be examined. The communication link between data and decision layers is the primary attack surface for SQL injection. This paper facilitates understanding what SQL injection is and why it is a significant threat to control system environments.



ACKNOWLEDGMENTS

This paper would not have been possible without the valuable research and analytic contributions provided by Kris Watts, University of Idaho scholar and Attack Methodology Analysis intern.



CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGMENTS	v
ACRONYMS.....	ix
1. INTRODUCTION.....	1
2. DATABASES AND SQL	2
3. UNDERSTANDING SQL INJECTION.....	4
4. HISTORY OF SQL INJECTION	5
5. FUTURE OF SQL INJECTION AS AN ATTACK METHODOLOGY	7



ACRONYMS

AMA	Attack Methodology Analysis
COTS	Commercial-Off-the Shelf
CS	Control System
DBMS	Database Management System
RDBMS	Relational Data Structures
SQL	Structure Query Language
XML	Extensible Markup Language



Attack Methodology Analysis: SQL Injection

1. INTRODUCTION

Threats to control system—To cut costs while increasing reliability and maintainability, modern control systems are using commercial off-the-shelf (COTS) applications, software development environments, and operating systems rather than the proprietary applications and protocols popular in legacy environments. From a security standpoint, this transition has both positive and negative implications for control system (CS) networks.

Consolidating to a narrower scope of applications, protocols, and operating systems that are well-known and well-supported in the general Information Technology (IT) world decreases the costs of system administration; this also allows for the implementation of standardized security practices and technologies that have been popularly tested and deployed by IT security administrators. The move away from non-standard, proprietary applications and protocols to COTS software facilitates the rapid exchange of information between CS environments and companies' business or financial networks, resulting in higher quality products and more competitive pricing for consumers.

However, this integration of COTS software into CS networks introduces a wide range of new computer-based threats to critical infrastructure. CS security previously depended on the principle of "security through obscurity." As long as each individual CS environment was highly customized and dependent on poorly documented, obscure proprietary protocols and applications, attackers required extensive knowledge of the CS network architecture, significant technical expertise, and hands-on experience with the CS application being used by the target. This is no longer the case.

Exploits and attack methodology, already widely used against the COTS software on IT networks, can be readily used or ported over for use against CS environments. This significantly eases the complexity of performing cyber attacks against control systems. The new level of threat faced by critical infrastructure is compounded by the move to incorporate communications—primarily from the CS databases—into organizations' business or financial networks, which in turn are connected to the public Internet. Now, overwhelming numbers of exploits exist for key components of CS networks and the vulnerabilities can be accessed via the web.

Proprietary application formats and obscure protocols no longer provide control systems with sufficient protection against cyber attacks. New threats to critical infrastructure computers must be identified as quickly as possible to ensure defenses can be tested and deployed rapidly and effectively. Databases used by control systems to store valuable data are often connected to databases or computers with web-enabled applications located on the business network. Virtually every data-driven application has transitioned to some form of database, and Structured Query Language (SQL) is the language used to query most of those databases.

Impact of SQL injection on control systems—The important information contained in these CS databases makes them high-value targets to attackers. By connecting CS databases to business or financial databases or computers with applications used to access the data, attackers can exploit the communications channel between the two networks and bypass the security mechanisms used to protect the CS environment. Since SQL injection is one of the primary forms of information gathering and exploitation used by attackers against databases, the evolution and significance of this particular attack methodology must be evaluated for its impact on CS security.

Injection into a database with valuable data can have far-reaching affects, especially in one such as a CS environment where data accuracy and integrity is critical for both business and operational decision-making. Control systems are more adversely affected by SQL injection than are many general IT databases because they are so reliant on data accuracy and integrity.

For example, a SQL compromise on a medical records database is problematic because of the data sensitivity and the cost of recovery. Data corruption or subversion of database information could cause CS network downtime, process control failures, or equipment failure in addition to network downtime, loss of profits, and cost of recovery. Successful compromise of a database could result in root compromise of the database and the computer it resides on. If the compromised system is a trusted host on a CS network, escalation of privileges on the CS network or on other CS components could easily be achieved, allowing attackers to directly manipulate the CS processes.

2. DATABASES AND SQL^a

General information—The most popularly-used databases in CS environments are IBM's DB2, Oracle, and Microsoft SQL Server; all of these allow queries and commands to be passed to the database via SQL. Common uses for the databases include: a) simple data storage, b) information linking, and c) credential authentication. The information contained within databases is generally accessed via front-end applications such as client/server or web applications that are user-friendly and facilitate data-gathering from the database in a fashion transparent to the end user.

Database components—Databases, in this paper, will refer to the Database Management System (DBMS) that may manage relational data structures (RDBMS), flat files, extensible markup language (XML) data, etc. The DBMS is the software which handles all aspects of data management including physical storage, reading and writing data, security, replication, error correction, and other functions. Each DBMS has a way of receiving a SQL statement and processing the instructions found in the statement. The DBMS carries out the instruction whether it is to modify a structure; to insert, update or delete a record; or to return a specific set of data back to the requester. Larger and more complex DBMS applications may be closely integrated with a specific operating system and some actually contain an OS optimized for data management. Some applications can also self-optimize after monitoring the actual conditions

^a *Beginning SQL Programming (Programmer to Programmer)*, by John Kauffman, et al., WROX, 2001.

after deployment. A DBMS may include additional features such as query analyzing, replication and back-up tools, user management, security tools, and performance monitoring.

SQL—SQL is a computer language for communication with databases. The communicating parties are typically a "front end" application that sends a SQL statement across a connection to a database that holds the data. SQL has many capabilities, but the most common needs in business are to:

- Read, analyze and report on existing data,
- Create new records holding data,
- Modify existing data,
- Append data to an existing record, and
- Delete data or records.

SQL contains key words or parts to perform these basic tasks. Reading data is the most common task. An ANSI-SQL statement requesting a list of names of all members of an organization can be sent from a Visual Basic application to an Oracle database. If the database is later changed to IBM's DB2, the SQL statement is still valid. The SQL language offers many permutations of the request, including the ability to return the names in various orders, only the first or last few names, a list of names without duplicates, and various other requests where people require specific information from their database.

Records can be created in a database using SQL. A data entry form on a web site can gather information from a visitor and then structure that data into a SQL statement. The SQL statement will instruct the DBMS to insert a new record into a database using the data gathered on the web page. Since SQL is universally accepted, the same SQL statement could, for example, be formed from, say, a Visual Basic application capturing the same information on a local network.

Data can also be changed using SQL. As in the examples above, a front-end user interface such as a web page can accept changes to data and send them via a SQL statement to the DBMS. But there does not have to be direct user interaction. A DB2 database running on an IBM mainframe could have a procedure to connect to another corporate mainframe running Sybase. DB2 can generate and send a SQL statement to modify the data in certain records in the Sybase database. Although the systems are from different vendors and have different ways of storing and using data, they both understand the SQL statement.

Deleting data can be performed using SQL statements. In fact, SQL can accommodate very complex sets of conditions for which records to delete and which to leave intact. Portions of data within a record can be deleted or entire tables within the database can be removed.

Note: Virtually every application that uses SQL to communicate with a database has expanded the protocol to include proprietary syntax and function calls. While syntax is typically consistent, some expansions of and enhancements to SQL facilitate SQL injection because of their design and implementation.

3. UNDERSTANDING SQL INJECTION

What is SQL injection—Attackers perform SQL injection in a number of ways, but there are a few basic steps that occur during any injection process. First, the attacker provides input into an application that passes the data over to the database. The malicious command is incorporated into a SQL query in some fashion that the front-end application screens and accepts as legitimate input, which includes giving the attacker some form of authentication and granting a level of privilege on the database.

Then, the application passes the user input and a command associated with that input field to the database, which interprets the user data according to the application command. Generally, the SQL syntax of the input data has been written in such a way that it adheres to the syntactical rules of the language but has some slight twist to the syntax. Finally, the database executes the application command with the interpreted data and produces some result such as database corruption, record deletion, or operating system compromise.

SQL injection example—The following is an example of “breaking out” of the quotation limitations per rainforestpuppy’s explanation. A full name fed to a MySQL database and encapsulated by quotes may still be used for injection.

Proper user input: john doe

*Translates to: SELECT * FROM records WHERE name="john doe";*

Malformed user input: john doe"; DROP TABLE records; #

*Translates to: SELECT * FROM records WHERE name="john doe"; Drop TABLE records;*

The trailing “#” in the “Malformed user input” command is the MySQL comment operator. It tells the MySQL database to ignore everything that follows the comment operator. This manipulation of SQL syntax allows the attacker to ensure the trailing quote is dropped and the query appears valid. If the program making the malformed query described above had sufficient privileges, the malicious input would destroy the entire records table. This attack could result in the modification or complete destruction of a database.

Significance of SQL injection as an attack methodology—Since SQL is highly standardized and is an open standard, it is well documented and attackers have detailed understanding of the technical parameters of the language and its intended use. The information also allows attackers to understand how SQL queries may be manipulated during attacks to exploit a database or to use error reports to enumerate the database structure, thereby facilitating the attacker’s ability to manipulate data and responses more effectively.

SQL is a language used to communicate with databases so it can be used by many different types of front-end applications to query the database. The most problematic of these are the web-

based front ends because users must be able to create, modify, and delete data. Essentially, the user accessing the front application via a website is given valid authentication token and high-level permissions on the database without having to demonstrate a valid need for them. The only safeguards in these applications are those included by the programmers to verify the user input, and given that secure code review is a relatively new practice, many front-end applications lack sufficient input checking to prevent SQL injection attacks.

The complexity of DBMS code and its focus on user functionality and data manipulation capacity also create increased opportunities for successful attack. The increasing integration of databases into web pages and other data-driven applications means developers need more functionality from the DBMS and expansions on the SQL language. As database vendors strive to improve their products to meet this need, they include more powerful control features for the programmers, making the impact of SQL injection more significant if those features are exploited. Additionally, programmers must understand how the advanced DBMS features impact the security of their applications, a time-consuming task at best. If the developers do not understand the security impact, the chances for software flaws that allow SQL injection to occur are greatly increased.

4. HISTORY OF SQL INJECTION

Initial SQL injection research—SQL injection as an attack method was first publicized as a side note to a comprehensive Microsoft web services exploitation article. The article first appeared in the fifty-fourth article of Phrack, a digital periodical that covers hacking topics. Titled “NT Web Technology Vulnerabilities,” the article was written by rainforestpuppy of the WireTrip security group, and discusses Microsoft SQL and ASP injection exploits.^b Rainforestpuppy approaches the injection technique as a side note to more serious vulnerabilities.

The Phrack 54 article served as a start point for SQL injection. An official advisory concerning the ability to batch commands was posted by the Allaire group (now part of Macromedia) several months later. The Allaire advisory, advisory number ASB99-04, generalized attack methods against ColdFusion, ASP, Sybase SQL, and Microsoft SQL applications.^c SQL injection was still a very new concept, and Allaire's advisory assumed that only a very narrow group of systems was vulnerable due to variables being encapsulated with quotes. Microsoft countered with the argument that the issues identified by rainforestpuppy were not vulnerabilities but they were features. The response prompted rainforestpuppy to aggressively pursue his research of SQL injection techniques.

The next exploitation and subsequent publication by rainforestpuppy proved Microsoft's claim to be very untrue. “How I hacked PacketStorm—a look at hacking www threads via SQL,”

^b “NT Web Technology Vulnerabilities,” rainforestpuppy, December 25, 1998, <http://www.phrack.org/show.php?p=54&a=8>

^c “Multiple SQL Statements in Dynamic Queries,” Allaire security bulletin, February 4, 1999, http://www.macromedia.com/devnet/security/security_zone/asb99-04.html

by rainforestpuppy, demonstrated how to directly circumvent some of the barriers to SQL injection that were assumed by the Allaire advisory.^d This article was the first to introduce a directed, successful attack using SQL injection and proved how easy it was to actively circumvent implied security features. The structure of the database was enumerated through random input and close analysis of the error reports generated by feeding the database the random data. Through his analysis of the database structure and his understanding of SQL query syntax, rainforestpuppy defeated the limits of quotation by “breaking out,” using additional sets of quotes to bypass SQL format constraints, and injecting his own commands into the database.

Using SQL injection as a primary exploitation tool—In early 2002, several papers were released detailing the use of built-in functions within most database applications. The functions in question allowed a user to issue commands from the database to the system. This marked the first time SQL injection was demonstrated as a method for operating system compromise.

An example of such a compromise involves the use of injection against a database resident on a Microsoft Windows operating system. The `xp_cmdshell` procedure allows applications to funnel information back and forth between applications and the operating system shell via a database. If an attacker could successfully inject a malicious command into the database using an account with execution permission, the attacker could then issue arbitrary commands to the shell. These commands would execute with system level, or administrative, permissions because the database typically ran as System, a privileged account, on Windows operating systems.

Current trends in research—Modern SQL injection research appears to be shifting from aggressive exploitation of hosts to using it as a means of discovery of vulnerabilities. Developers have become more familiar with SQL injection techniques and have begun to attempt to hide revealing information, validate user input more stringently, and limit attackers’ access to SQL injection points. New versions of scripting languages like PHP and Perl include variable filtering of SQL input. These defensive techniques have made exploitation of hosts via SQL injection more difficult.

While actual query exploitation has remained relatively unchanged during the past two years, discovery and mapping of vulnerable hosts has changed drastically. Functions such as `sendmail` are being employed to subvert error suppression. For example, if an application suppresses all signs of error or success via the front end application, an attacker may be able to utilize `sendmail` functionality in the DBMS to mail the results to him or herself.

From early 2002 to mid-2004, a number of papers were published that focused primarily on a form of database enumeration known as blind SQL injection. Blind SQL injection is the act of mining information regarding the layout of database structures and the feasibility of injection when error messages are suppressed. A skilled attacker can map out a vulnerable application through carefully sequenced syntactical inputs, looking for changes in server behavior. An application that reacts differently to database errors than application level errors provides

^d “How I hacked PacketStorm,” rainforestpuppy, February 3, 2000, <http://cert.uni-stuttgart.de/archive/bugtraq/2000/02/msg00082.html>

confirmation of a successful query. The confirmation alone is enough information to begin making slight query variations to determine how input is passed to the database and what functions are available.

BlackHat 2004, a well-known information security research conference, brought the release of the first publicly available tool for automating the SQL injection process. Absinthe (formerly known as SqueaL) was released by the security group 0x90.org and greatly trivialized the process of gathering information about the structure of a database. Absinthe changed SQL-style injections as it proved that attacks on applications through injection could become automated. Automation changed the way attacks were carried out, shifting attack methodologies from localized and directed to widespread and ambiguous, with the possibility of worm-like behaviors. Security techniques that relied on error suppression and general obscurity had to be completely rethought. Applications that had previously avoided exploitation were appearing on vulnerability lists at an increasingly rapid pace.

5. FUTURE OF SQL INJECTION AS AN ATTACK METHODOLOGY

Level of interest by researchers—The names of those associated with innovative SQL injection research are typically available in only two forms: publications or arrest records. The groups and individuals who have published white papers regarding SQL injection represent both white hat and black hat security groups.

Big name security firms such as NGS Software have led the research in SQL injection attacks, pioneering enumeration and exploitation techniques of all the major databases via innovative SQL manipulation. These firms have made a great deal of money consulting with database companies and will continue to do so as long as they can find security concerns with them.

The black hat community has had resounding success with SQL injection and will most certainly keep developing innovative new techniques in the field. SQL injection will remain a standard tool in penetration testing and exploit toolkits, much like packet fragmentation and buffer overflows, as long as poorly coded applications or legacy systems still remain.

Defensive responses—Fortunately, as the popularity of SQL injection has grown, Oracle, Microsoft, and IBM have begun to evaluate their database code for security issues as have the other database vendors. They have begun to incorporate security into their product line and are moving more quickly to resolve security issues associated with SQL injection-based attacks. Database and systems administrators harden both the database and the operating system as much as possible prior to deploying the system for production use. The programmers responsible for writing major enterprise applications have begun to review user input more closely and incorporating more stringent security techniques into the code, making new versions of their software more resistant to the known forms of SQL injection. Scripting languages such as PHP now include variable content filtering that scrubs SQL queries before they are sent to the database.

These defensive responses have evolved in response to the IT security community's recognition of the seriousness of the SQL injection threat. The number of critical vulnerabilities announced by vendors because of SQL injection exploits has decreased greatly in most widely used enterprise applications. Many of the SQL injection vulnerabilities are in niche applications, whose software developers may not fully understand the ramifications of insecure coding, because major vendors have made SQL injection more difficult to perform.

Unfortunately, this still leaves a large number of vulnerable systems open for attack. New applications whose programmers did not incorporate sufficient security mechanisms, outdated applications that are not slated for upgrade because of financial issues, and niche applications that will not run in secured environments all hinder security efforts and are commonly found on both IT and CS networks. SQL injection will remain a popular form of attack until vulnerable applications can be replaced. The timeline for such a change in the networked computing environment will last several years. Since software upgrades are costly, security enhancements happen slowly, leaving attackers a significant period of time during which SQL injection will prove effective for them to exploit networks hosting SQL databases.

Impact on control system security—CS networks have not been designed with security as an inherent feature in the system. Rather, security is an added layer on the network, not a core component of network communication design, operating system administration, or application development. Given the reliance of CS systems on the storage, accuracy, and accessibility of CS data and the prevalence of SQL databases on these networks, standard SQL injection techniques against CS components using COTS database technology and customized SQL injection exploits designed to run against proprietary CS applications pose a major threat to CS security.

Allocation of defensive resources—As the IT community begins to become more familiar with SQL injection and incorporate defensive technologies to limit attacks, these methods can be evaluated for use in CS networks. CS security should include:

- Securing communications between business applications and CS databases;
- Incorporating design techniques within the database to circumvent injection attempts;
- Increasing the defenses on CS components containing databases or layering security around those components if they cannot be hardened;
- Ensuring that adequate means of detecting SQL injection attacks against CS databases are in place; and,
- Purchasing new products that actively incorporate SQL security into database and application design when legacy systems are replaced.